On Finding Multi-constrained Paths *

Shigang Chen†, Klara Nahrstedt
Department of Computer Science
University of Illinois at Urbana-Champaign
{s-chen5, klara}@cs.uiuc.edu

Abstract

New emerging distributed multimedia applications provide guaranteed end-to-end quality of service (QoS) and have stringent constraints on delay, delay-jitter, cost, etc. The task of QoS routing is to find a route in the network which has sufficient resources to satisfy the constraints. Multi-constrained routing is difficult because different constraints can conflict with one another. In particular, the delay-cost-constrained routing problem is NP-complete.

We propose a heuristic algorithm for this problem. The idea is to first reduce the NP-complete problem to a simpler one which can be solved in polynomial time, and then solve the new problem by either an extended Dijkstra's algorithm or an extended Bellman-Ford algorithm. When the extended Dijkstra's algorithm is used, the total time complexity of the heuristic algorithm is $O(x^2V^2)$; when the extended Bellman-Ford algorithm is used, the total complexity is $O(xVE)$, where x is an integer defined solely by the algorithm. We prove the correctness of our algorithm by showing that a solution to the simpler problem must also be a solution to the original problem. The performance of the algorithm can be adjusted by tuning the value of x. A larger value for x results in a higher probability of finding a solution and a higher overhead.

1 Introduction

Quality of Service (QoS) routing has been attracting considerable attention in the research community recently [5, 9, 10, 11, 12]. The routing requests are typically specified in terms of constraints.

*This work was supported by the ARPA grant under contract number F30602-97-2-0121 and the National Science Foundation Career grant under contract number NSF CCR 96-23867.

†Please address all correspondences to Shigang Chen at Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 61801, phone: (217) 244 5841.

For example, a bandwidth constraint requires the bandwidth of a routing path to be not less than a given lower bound, and a delay (cost) constraint requires the total delay (cost) of a path to be not greater than a given upper bound. The multi-constrained routing problem is difficult because different constraints can conflict with one another. In particular, the delay-cost-constrained routing, i.e., finding a route between two nodes in the network with both end-to-end delay and end-to-end cost bounded, can be formalized as a *multi-constrained path problem* (MCP), which is NP-complete [4, 12].

Many existing QoS routing algorithms [9, 10, 11] consider a single constraint. The competitive routing strategy discussed in [9] considers only the bandwidth constraint. The Salama's algorithm [10] and the Sun's algorithm [11] consider the delay constraint, while using heuristic approaches to minimize the cost of the found route. Some multi-constrained routing algorithms work for special cases where the constraints are not independent. For example, in the Ma's algorithm [5], delay, delay-jitter and buffer space are all functions of bandwidth, which makes the problem solvable in polynomial time. Jaffe [4] proposed a distributed algorithm to solve the NP-complete MCP problem in pseudo-polynomial time. A more in-depth discussion of the related work can be found in Section 3.

We propose a heuristic algorithm for the MCP problem with a polynomial time complexity. The idea is to first reduce the NP-complete problem to a simpler problem which can be solved in polynomial time by an extended Dijkstra's or Bellman-Ford algorithm, and then solve the simpler problem to find a solution path. We prove that a solution path for the simpler problem is also a solution path for the original problem. The performance of the algorithm is predictable and adjustable. It is predictable in the sense that when certain condition is satisfied the algorithm is guaranteed to find a solution. It is adjustable in the sense that the probability of finding a solution can be increased when a special integer value defined by the algorithm is increased.

The rest of the paper is organized as follows. In Section 2, the heuristic algorithm with two constraints is presented first; it is then generalized for an arbitrary number of constraints; the delay-cost-constrained routing algorithm is discussed next; finally, the performance of the routing algorithm is studied by experiments. The related work is covered in Section 3. Section 4 draws the conclusion.

2 A Polynomial-time Heuristic algorithm

2.1 The heuristic algorithm

Let $\mathbf{R}_0^+$ be the set of non-negative real numbers and $\mathbf{I}$ the set of non-negative integers.

Definition 1 *Multi-constrained path problem* (MCP): Given a directed graph $G\langle V, E\rangle$, a source vertex s, a destination node t, two weight functions $w_1 : E \rightarrow \mathbf{R}_0^+$ and $w_2 : E \rightarrow \mathbf{R}_0^+$, two constants $c_1 \in \mathbf{R}_0^+$ and $c_2 \in \mathbf{R}_0^+$; the problem, denoted as MCP$(G, s, t, w_1, w_2, c_1, c_2)$, is to find a path p from s to t such that $w_1(p) \leq c_1$ and $w_2(p) \leq c_2$ if such a path exists.

A path p which satisfies $w_1(p) \leq c_1$ and $w_2(p) \leq c_2$ is called a *solution* to MCP$(G, s, t, w_1, w_2, c_1, c_2)$. We assume that both weight functions are *additive* — the weight of a path is equal to the summation of the weights of all edges on the path.

Definition 2 *w_1-weight and w_2-weight*: For a path $p = v_0 \rightarrow v_1 \rightarrow ..., \rightarrow v_k$,

$$w_1(p) = \sum_{i=1}^{k} w_1(v_{i-1}, v_i) \quad \text{and} \quad w_2(p) = \sum_{i=1}^{k} w_2(v_{i-1}, v_i)$$

$w_1(p)$ is called the *w_1-weight* and $w_2(p)$ the *w_2-weight* of the path p.

MCP$(G, s, t, w_1, w_2, c_1, c_2)$ is NP-complete [12]. We provide a polynomial-time heuristic solution to this problem. The algorithm contains two steps:

1. Create a new weight function $w_2' : E \rightarrow I$.

$$w_2'(u, v) = \left\lceil \frac{w_2(u, v) \cdot x}{c_2} \right\rceil \tag{1}$$

   where x is a given positive integer. We reduce the original problem MCP$(G, s, t, w_1, w_2, c_1, c_2)$ to a new, simpler problem MCP$(G, s, t, w_1, w_2', c_1, x)$. [1]

2. Solve MCP$(G, s, t, w_1, w_2', c_1, x)$ in polynomial time.

The algorithms for Step 2 will be discussed in Section 2.2. We assume for the moment that a solution of MCP$(G, s, t, w_1, w_2', c_1, x)$ can be found in polynomial time if there exists one.

[1] Note that the value of x is chosen by the algorithm. It does not depend on the input values of G, s, w_1, w_2, c_1 and c_2. This is the essential reason for Step 2 of the algorithm to be solvable in polynomial time. As we will see shortly, a larger x means a higher chance to find a satisfactory path and a higher overhead.

Since MCP$(G, s, t, w_1, w_2, c_1, c_2)$ is NP-complete, we are not trying to find a solution for it whenever there exists one. The idea is to reduce it to a simpler and solvable problem, MCP$(G, s, t, w_1, w_2', c_1, x)$, which has a "coarser resolution" — x is a given finite integer and the range of w_2' is **I**. Theorem 1 guarantees that a solution to the simpler problem must be a solution to the original problem.

Theorem 1 A solution to MCP$(G, s, t, w_1, w_2', c_1, x)$ must also be a solution to MCP$(G, s, t, w_1, w_2, c_1, c_2)$.

Proof: Let p be a solution to MCP$(G, s, t, w_1, w_2', c_1, x)$. Hence, $w_1(p) \leq c_1$ and $w_2'(p) \leq x$. In order to prove p is also a solution to MCP$(G, s, t, w_1, w_2, c_1, c_2)$, it suffices to prove $w_2(p) \leq c_2$. By the definition of w_2', we have

$$
\begin{aligned}
w_2'(u,v) &= \left\lceil \frac{w_2(u,v) \cdot x}{c_2} \right\rceil \qquad \text{by (1)} \\
w_2'(u,v) &\geq \frac{w_2(u,v) \cdot x}{c_2} \\
w_2(u,v) &\leq \frac{w_2'(u,v) \cdot c_2}{x}
\end{aligned}
$$

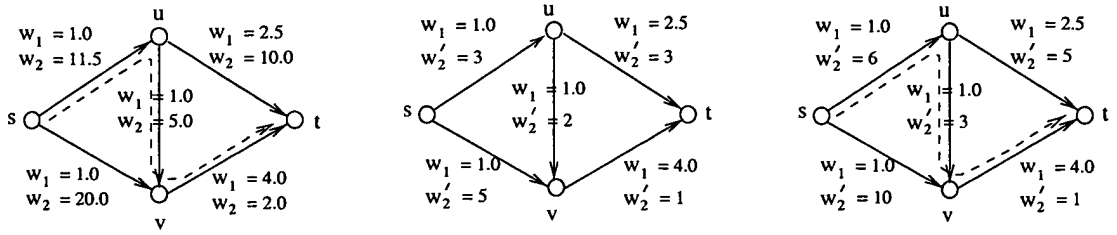
Therefore, we have

$$
\begin{aligned}
w_2(p) &= \sum_{(u,v) \text{ on } p} w_2(u,v) \\
&\leq \sum_{(u,v) \text{ on } p} \frac{w_2'(u,v) \cdot c_2}{x} \\
&= \frac{c_2}{x} \cdot \sum_{(u,v) \text{ on } p} w_2'(u,v) \\
&= \frac{c_2}{x} \cdot w_2'(p) \\
&\leq \frac{c_2}{x} \cdot x \\
&= c_2
\end{aligned}
$$

$w_2(p) \leq c_2$ and hence the theorem holds. □

Corollary 1 Let P be the set of solutions to MCP$(G, s, t, w_1, w_2, c_1, c_2)$ and P' be the set of solutions to MCP$(G, s, t, w_1, w_2', c_1, x)$. Then,

$$P' \subseteq P$$



(a) MCP(G, s, t, w_1, w_2, 8.0, 20.0) (b) MCP(G, s, t, w_1 , w_2', 8.0, 5) (c) MCP(G, s, t, w_1, w_2', 8.0, 10)

Figure 1: (a) The path shown by the dotted line, $s \rightarrow u \rightarrow v \rightarrow t$, is a solution to MCP$(G, s, t, w_1, w_2, 8.0, 20.0)$. (b) Consider $x = 5$. The problem is reduced to MCP$(s, t, w_1, w_2', 8.0, 5)$, which does not have a solution. (c) If x is increased to 10, the problem is reduced to MCP$(s, t, w_1, w_2', 8.0, 10)$, which has a solution, $s \rightarrow u \rightarrow v \rightarrow t$. Note that w_2' in (c) is different from that in (b) because the values of x are different.

Corollary 2 Let P' be the set of solutions to MCP$(G, s, t, w_1, w_2', c_1, x)$. The heuristic algorithm succeeds in finding a solution to MCP$(G, s, t, w_1, w_2, c_1, c_2)$ if and only if $P' \neq \emptyset$.

The converse of Theorem 1 is not necessarily true — a solution to MCP$(G, s, t, w_1, w_2, c_1, c_2)$ may not be a solution to MCP$(G, s, t, w_1, w_2', c_1, x)$. Figure 1 gives an example. The original problem MCP$(G, s, t, w_1, w_2, 8.0, 20.0)$ has a solution $s \rightarrow u \rightarrow v \rightarrow t$ (Figure 1 (a)) . Suppose $x = 5$ and the problem is reduced to MCP$(G, s, t, w_1, w_2', 8.0, 5)$ (Figure 1 (b)). The path $s \rightarrow u \rightarrow v \rightarrow t$ is not a solution to the new problem. In fact, there is no solution to the new problem.

Hence, our heuristic algorithm may not find a solution for MCP$(G, s, t, w_1, w_2, c_1, c_2)$ even when such a solution exists, because the solution set P' of the new problem MCP$(G, s, t, w_1, w_2', c_1, x)$ can be empty. Fortunately, whether P' is empty or not is to some extend predictable and adjustable — by assigning a larger x, we have a better chance for P' to be non-empty.

Theorem 2 [2] Let a path p be a solution to MCP$(G, s, t, w_1, w_2, c_1, c_2)$ and l is the length of p. If

$$w_2(p) \leq (1 - \frac{l-1}{x}) \cdot c_2 \tag{2}$$

then p is also a solution to MCP$(G, s, t, w_1, w_2', c_1, x)$.

[2] Theorem 2 can be rewritten as: A path p from s to t is a solution to MCP$(G, s, t, w_1, w_2', c_1, x)$ if

$$w_1(p) \leq c_1 \ \wedge \ w_2(p) \leq (1 - \tfrac{l-1}{x}) \cdot c_2$$

Proof: Since p is a solution to MCP$(G, s, t, w_1, w_2, c_1, c_2)$, we already have $w_1(p) \leq c_1$. In order to prove p is a solution to MCP$(G, s, t, w_1, w_2', c_1, x)$, we only need to prove $w_2'(p) \leq x$.

$$
\begin{aligned}
w_2'(p) &= \sum_{(u,v) \text{ on } p} w_2'(u,v) \\
&= \sum_{(u,v) \text{ on } p} \left\lceil \frac{w_2(u,v) \cdot x}{c_2} \right\rceil && \text{by (1)} \\
&< \sum_{(u,v) \text{ on } p} \left(\frac{w_2(u,v) \cdot x}{c_2} + 1 \right) \\
&= \frac{x}{c_2} \cdot \sum_{(u,v) \text{ on } p} w_2(u,v) + \sum_{(u,v) \text{ on } p} 1 \\
&= \frac{x}{c_2} \cdot w_2(p) + l \\
&\leq \frac{x}{c_2} \cdot \left(1 - \frac{l-1}{x}\right) \cdot c_2 + l && \text{by (2)} \\
&= (x - l + 1) + l \\
&= x + 1
\end{aligned}
$$

Because both $w_2'(p)$ and x are integers, $w_2'(p) \leq x$. Therefore, the theorem holds. □

Theorem 2 means that if there exists a path p which is *overqualified* — not just $w_2(p) \leq c_2$ but $w_2(p) \leq (1 - \frac{l-1}{x}) \cdot c_2$ — then after we reduce the original problem to MCP$(G, s, t, w_1, w_2', c_1, x)$, the new problem still has solutions (p is one of them). Hence, we can solve MCP$(G, s, t, w_1, w_2', c_1, x)$ to find a solution, which must also be a solution to the original problem as stated by Theorem 1.

Corollary 3 Let P be the set of solutions to MCP$(G, s, t, w_1, w_2, c_1, c_2)$ and P' be the set of solutions to MCP$(G, s, t, w_1, w_2', c_1, x)$. Then,

$$P' \neq \emptyset \text{ if } P \neq \emptyset \text{ and } \exists p \in P, w_2(p) \leq \left(1 - \frac{l-1}{x}\right) \cdot c_2$$

Theorem 3 [3] Let P be the set of solutions to MCP$(G, s, t, w_1, w_2, c_1, c_2)$. The heuristic algorithm succeeds in finding a solution to MCP$(G, s, t, w_1, w_2, c_1, c_2)$ if

$$P \neq \emptyset \text{ and } \exists p \in P, w_2(p) \leq (1 - \frac{l-1}{x}) \cdot c_2$$

Proof: By combining Corollaries 2 and 3. □

Note that, by Theorem 3, if $P \neq \emptyset$ and $\exists p \in P, w_2(p) \leq (1 - \frac{l-1}{x})c_2$, then our heuristic algorithm is *guaranteed* to find a solution. However, even when $P \neq \emptyset$ and $\forall p \in P, w_2(p) > (1 - \frac{l-1}{x})c_2$, the algorithm still has a good chance to find one, though that is not guaranteed. Figure 1 gives an example. In Figure 1 (a), MCP$(G, s, t, w_1, w_2, 8.0, 20.0)$ has only one solution, which is $p = s \rightarrow u \rightarrow v \rightarrow t$. $w_2(p) = 19.5$. Consider $x = 10$ in Figure 1 (c). The problem is reduced to MCP$(G, s, t, w_1, w_2', 8.0, 10)$. Because $l = 3$ and $c_2 = 20.0$, $(1 - \frac{l-1}{x})c_2 = 18.0$. Hence, the condition $w_2(p) \leq (1 - \frac{l-1}{x})c_2$ is not satisfied. However, p is still a solution to the new problem MCP$(G, s, t, w_1, w_2', 8.0, 10)$. We thus conclude that $w_2 \leq (1 - \frac{l-1}{x})c_2$ is a sufficient but not a necessary condition for a solution of the original problem to remain as a solution of the new problem.

How restrictive is the condition $w_2(p) \leq (1 - \frac{l-1}{x})c_2$? Suppose a distributed multimedia application wants a connection in a network with delay no more than 300ms and delay jitter no more than 100ms. Suppose there exists a path with a length of 5, a delay bound of 300ms and a delay-jitter bound of 90ms. Then with $x = 40$ our algorithm will definitely find a path for the application.

Let us consider the worst case where the longest loop-free path in G is $|V|$. Table 1 shows the conditions under various different values of x. A larger x corresponds to a more relaxed condition, which implies a better chance to find a solution and a higher overhead — as we shall see in Section 2.2, the overhead is proportional to x when an extended Bellman-Ford algorithm is used to implement Step 2 of the heuristic algorithm. Take the case $x = 10|V|$ as an example. The corresponding condition is $w_2(p) \leq 0.9c_2$. It means that, given an arbitrary problem MCP$(G, s, t, w_1, w_2, c_1, c_2)$, if the problem has a solution p such that $w_2(p) \leq 0.9c_2$, then our heuristic algorithm is *guaranteed* to find a solution for the problem, provided x is as large as $10|V|$.

[3] Theorem 3 can be rewritten as: The heuristic algorithm succeeds in finding a solution to MCP$(G, s, t, w_1, w_2, c_1, c_2)$ if there exists a path p from s to t such that

$$w_1(p) \leq c_1 \ \wedge \ w_2(p) \leq (1 - \frac{l-1}{x}) \cdot c_2$$

x	$w_2(p) \leq (1 - \frac{l-1}{x}) \cdot c_2$
$2\|V\|$	$w_2(p) \leq 0.5c_2$
$5\|V\|$	$w_2(p) \leq 0.8c_2$
$10\|V\|$	$w_2(p) \leq 0.9c_2$
$100\|V\|$	$w_2(p) \leq 0.99c_2$
$+\infty$	$w_2(p) \leq c_2$

Table 1: x v.s. $w_2 \leq (1 - \frac{l-1}{x}) \cdot c_2$

For real applications, our algorithm is expected to perform much better than what stated in Table 1. Consider the QoS routing problem by which our heuristic algorithm is originally motivated. The lengths of the routing paths between two nodes in the network should be much less than $|V|$. Therefore, to achieve a condition such as $w_2(p) \leq 0.9c_2$, the value of x can be far less than $10|V|$, which means the actual overhead of the algorithm can be much lower than that of the worst case. Since the shortest paths are often preferred to save resources, a practical value for x can be $10d_{s,t}$, where $d_{s,t}$ is the distance between s and t.

One important question still remains: Is the new problem in Step 2 solvable in polynomial time? We answer this question in Section 2.2.

2.2 The extended Dijkstra's and Bellman-Ford algorithms

The problem MCP$(G, s, t, w_1, w_2', c_1, x)$, where $w_2' : E \in \mathbf{I}$, is solvable in polynomial time for a given $x \in \mathbf{I}$. Note that no restriction has been placed on G, s, w_1 and c_1, which can take any values allowed by Definition 1.

An extended Dijkstra's shortest path algorithm (EDSP) and an extended Bellman-Ford algorithm (EBF) are presented in Figure 2 to solve MCP$(G, s, t, w_1, w_2', c_1, x)$. An algorithm similar to EBF, in its distributed implementation, has been proposed by Jaffe [4]. We will discuss the difference between our algorithm and the Jaffe's algorithm in Section 3.

For each vertex $v \in V$ and each integer $k \in [0..x]$, a variable $d[v, k]$ is maintained, which is an **estimation** of the smallest w_1-weight of those paths from s to v whose w_2'-weights are k. Let

$$\delta(v, k) = \underset{p \in P(v,k)}{Min} \{w_1(p)\}$$

where $P(v, k) = \{p \mid w_2'(p) = k,\ p \text{ is a path from } s \text{ to } v\}$. The value of $d[v, k]$, initially $+\infty$, is

```
    Initialize(G, s)
    begin
(1)     for each vertex v ∈ V[G], each i ∈ [0..x] do
(2)         d[v, i] := ∞
(3)         π[v, i] := NIL
(4)     for each i ∈ [0..x] do
(5)         d[s, i] := 0
    end

    Relax(u, k, v)
    begin
(6)     k' := k + w'_2(u, v)
(7)     if k' ≤ x then
(8)         if d[v, k'] > d[u, k] + w_1(u, v) then
(9)             d[v, k'] := d[u, k] + w_1(u, v)
(10)            π[v, k'] := u
    end

    EDSP(G, s)
    begin
(11)    Initialize(G, s)
(12)    S := ∅
(13)    Q := {⟨u, k⟩ | u ∈ V[G], k ∈ [0..x]}
(14)    while Q ≠ ∅ do
(15)        find ⟨u, k⟩ ∈ Q such that d[u, k] = Min_{⟨u',k'⟩∈Q} {d[u', k']}
(16)        Q := Q − {⟨u, k⟩}
(17)        S := S + {⟨u, k⟩}
            /* Note that the for loop iterates on different v's with fixed values for u and k.
                The values of u and k are changed by the outer while loop. */
(18)        for each outgoing edge of u, (u, v) ∈ E do
(19)            Relax(u, k, v)
    end

    EBF(G, s)
    begin
(20)    Initialize(G, s)
(21)    for i := 1 to |V[G]| − 1 do
(22)        for k := 0 to x do
(23)            for each edge (u, v) do
(24)                Relax(u, k, v)
    end
```

Figure 2: The Extended Dijkstra's Shortest Path algorithm (EDSP) and the Extended Bellman-Ford algorithm (EBF)

always greater than or equal to $\delta(v,k)$. During the execution, EDSP (EBF) makes better and better estimation and $d[v,k]$ becomes closer and closer to, and eventually reach, $\delta(v,k)$.

When EDSP (EBF) completes, $d[v,k] = \delta(v,k), v \in V, k \in [0..x]$. There exists a solution, i.e. a path p from s to t such that $w_1(p) \leq c_1$ and $w_2'(p) \leq x$, iff $\exists k \in [0..x]$, $d[t,k] \leq c_1$. The path is stored by π. The variable $\pi[t,k]$ keeps the immediate preceding vertex (called *predecessor*) of t on the path. Hence, the path can be recovered by tracing the variable π of all intermediate vertices till reaching the source s.

Two additional variables, S and Q, are required by EDSP.

$$S = \{\langle v,k\rangle \mid d[v,k] = \delta(v,k), v \in V, k \in [0..x]\}$$

$$Q = \{\langle v,k\rangle \mid d[v,k] > \delta(v,k), v \in V, k \in [0..x]\}$$

where the notation $\langle v,k\rangle$ simply means a pair of values, $v \in V$ and $k \in [0..x]$. Initially, $S = \emptyset$ and $Q = \{\langle v,k\rangle | v \in V, k \in [0..x]\}$. By the **while** loop (lines 14-19 in Figure 2), each iteration moves a pair from Q to S and adjusts the w_1-weight estimation by calling Relax(u,k,v). When $Q = \emptyset$, the algorithm completes.

A more detailed presentation of the original Dijkstra's and Bellman-Ford algorithms, which our algorithms are based on, can be found in [2].

The time complexity of ESDP is (x^2V^2). The maximum size of Q is $(x+1)V$. Hence, line 15 can be done within $O(xV)$. There can be at most $(x+1)V$ iterations of the **while** loop and thus the total time for line 15 is $O(x^2V^2)$. The **for** loop of lines 18-19 has $(x+1)E$ iterations in total [4] because Relax(u,k,v) is called once for every $(u,v) \in E$, $k \in [0..x]$. In each iteration, Relax(u,k,v) takes $O(1)$. Hence, the time complexity for this part is $O(xE)$. The total time complexity is $O(x^2V^2 + xE) = O(x^2V^2)$. The time complexity of EBF is $O(xVE)$, because line 23 is executed for at most $(x+1)(V-1)E$ times. The space complexities of both algorithms are $O(xV)$.

Let us consider the time complexity of our heuristic algorithm in Section 2.1. Step 1 of the algorithm takes $O(E)$. Step 2 of the algorithm is implemented by EDSP or EBF. Therefore, the total time complexity is $O(x^2V^2)$ when EDSP is used or $O(xVE)$ when EBF is used. The time complexity is polynomial because the value of x is given by the algorithm. For example, if we let $x = 10|V|$ and use EBF in Step 2, the time complexity is $O(V^2E)$.

[4] $(x+1)E$ iterations are the combination result of the outer **while** loop and the inner **for** loop. The **while** loop iterates on u and k, and the **for** loop iterates on v.

2.3 Algorithms for more than two constraints

We have been studying MCP with two constraints so far. In this section, we shall generalize the algorithms for more than two constraints. We define MCP$(G, s, t, w_1, ..., w_n, c_1, ..., c_n), n \geq 2$, as to find a path p from s to t such that $w_i \leq c_i$, for each $i \leq n$.

The heuristic algorithm is generalized to be:

1. For each $i \in [2..n]$, create a new weight function $w_i' : E \rightarrow I$.

$$w_i'(u,v) = \left\lceil \frac{w_i(u,v) \cdot x_i}{c_i} \right\rceil$$

   where x_i is a given positive integer. We reduce the original problem MCP$(G, s, t, w_1, ..., w_n, c_1, ..., c_n)$ to a new, simpler problem MCP$(G, s, t, w_1, w_2', ..., w_n', c_1, x_2, ..., x_n)$, which is to find a path p such that $w_1(p) \leq c_1$ and $w_i'(p) \leq x_i, 2 \leq i \leq n$.

2. Solve MCP$(G, s, t, w_1, w_2', ..., w_n', c_1, x_2, ..., x_n)$ by EDSP or EBF.

The generalized EDSP and EBF can be found in Figure 3.

The time complexities of EDSP and EBF are $O(x_1^2 \cdot ... \cdot x_n^2 V^2)$ and $O(x_1 \cdot ... \cdot x_n VE)$, respectively. For sparse graphs in which $E = O(V)$, EBF is likely to perform better than EDSP, especially when n is large. The time complexity of the generalized heuristic algorithm equals that of EDSP or EBF, depending on which is used for Step 2.

2.4 Multi-Constrained Routing

Multi-Constrained routing is an important application of MCP. A network can be modeled as a graph $G\langle V, E\rangle$. V is the set of routing nodes and E is the set of communication links. Each link has two properties: *delay* and *cost*. [5] The delay (cost) of a path is the summation of the delays (costs) of all links on the path. Given a source node s and a destination node t, the multi-constrained routing problem is to find a path p from s to t such that $delay(p) \leq D$ and $cost(p) \leq C$, where D and C are the required end-to-end delay bound and cost bound, respectively. This is clearly a MCP problem. The routing algorithm is presented below.

[5] The cost of an edge can be measured in dollars, or it can be a function of a given system metric such as bandwidth utilization or buffer utilization.

```
Initialize(G, s)
begin
    for each vertex v ∈ V[G], each k_i ∈ [0..x_i], 2 ≤ i ≤ n do
        d[v, k_2, ..., k_n] := ∞
        π[v, k_2, ..., k_n] := NIL
    for each k_i ∈ [0..x_i], 2 ≤ i ≤ n do
        d[s, k_2, ..., k_n] := 0
end

Relax(u, k_2, ..., k_n, v)
begin
    for each i ∈ [2..n] do
        k'_i := k_i + w'_i(u, v)
    if k'_i ≤ x_i, for each i ∈ [2..n] then
        if d[v, k'_2, ..., k'_n] > d[u, k_2, ..., k_n] + w_1(u, v) then
            d[v, k'_2, ..., k'_n] := d[u, k_2, ..., k_n] + w_1(u, v)
            π[v, k'_2, ..., k'_n] := u
end

EDSP(G, s)
begin
    Initialize(G, s)
    S := ∅
    Q := {⟨u, k_2, ..., k_n⟩ | u ∈ V[G], k_i ∈ [0..x_i], 2 ≤ i ≤ n}
    while Q ≠ ∅ do
        find ⟨u, k_2, ..., k_n⟩ ∈ Q such that d[u, k_2, ..., k_n] = Min_{⟨u', k'_2, ..., k'_n⟩ ∈ Q} {d[u', k'_2, ..., k'_n]}
        Q := Q − {⟨u, k_2, ..., k_n⟩}
        S := S + {⟨u, k_2, ..., k_n⟩}
        for each edge (u, v) ∈ E do
            Relax(u, k_2, ..., k_n, v)
end

EBF(G, s)
begin
    Initialize(G, s)
    for i := 1 to |V[G]| − 1 do
        for each k_i ∈ [0, x_i], 2 ≤ i ≤ n do
            for each edge (u, v) do
                Relax(u, k_2, ..., k_n, v)
end
```

Figure 3: The generalized EDSP and EBF

1. Create two new functions $new\text{-}delay : E \rightarrow I$ and $new\text{-}cost : E \rightarrow I$.

$$new\text{-}delay(u,v) = \left\lceil \frac{delay(u,v) \cdot x}{D} \right\rceil$$

$$new\text{-}cost(u,v) = \left\lceil \frac{cost(u,v) \cdot x}{D} \right\rceil$$

where $x = coef \times d_{s,t}$, $coef$ is a given positive integer and $d_{s,t}$ is the distance from s to t. We reduce the original problem MCP$(G, s, t, delay, cost, D, C)$ to two simpler problems, MCP$(G, s, t, delay, new\text{-}cost, D, x)$ and MCP$(G, s, t, new\text{-}delay, cost, x, C)$.

2. First, solve MCP$(G, s, t, delay, new\text{-}cost, D, x)$ by EDSP or EBF. If a solution is found, return the found path and terminate; otherwise, solve MCP$(G, s, t, new\text{-}delay, cost, x, C)$.

A distributed implementation of the above algorithm is possible. However, without further simplification of the algorithm, the number of messages to be sent is expected to be very high [4]. Hence, we assume a source routing strategy, which was also adopted by routing algorithms in [6, 5, 8, 12]. It requires every node to maintain the state information of the network, which can be done by the link-state algorithm [7]. The routing path is determined locally at the source node.

The proposed routing algorithm applies the heuristic algorithm (Section 2.1) twice, reducing *delay* and *cost* to *new-delay* and *new-cost*, respectively. Hence, it guarantees to find a solution when *either* of the following two conditions is satisfied by a path p from s to t (see Theorem 3):

1. $delay(p) \leq (1 - \frac{l-1}{x}) \cdot D \ \wedge \ cost(p) \leq C$ (Heuristic condition one)

2. $delay(p) \leq D \ \wedge \ cost(p) \leq (1 - \frac{l-1}{x}) \cdot C.$ (Heuristic condition two)

The above two conditions are called the *heuristic conditions*. Note that we use the heuristic conditions to theoretically study the performance of the algorithm but they are *not* part of the algorithm.

The heuristic conditions are sufficient conditions in the sense that if one of them is satisfied our proposed routing algorithm *guarantees* to find a solution — it makes the performance of the algorithm predictable to some extend. However, as we discussed in Section 2.1 and showed by the example in Figure 1 (c), they are not necessary conditions — even when none of the heuristic conditions is satisfied, our algorithm may still find a solution. Hence, the heuristic conditions give a pessimistic estimation (or lower bound) of the performance of the proposed routing algorithm.

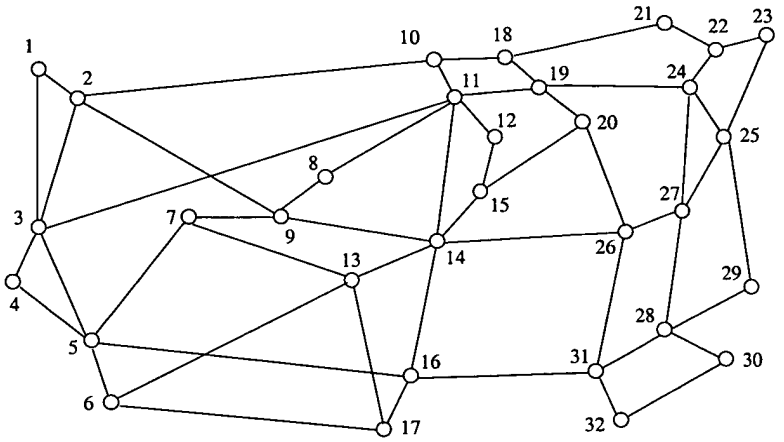


Figure 4: the topology of the experimental network

2.5 Experiments

We know from the routing algorithm proposed in Section 2.4 that $x = coef \times d_{s,t}$. What is the relationship between $coef$ and the performance of the algorithm and how large should $coef$ be? We answer the questions by simulation.

The network topology used in our simulation is shown in Figure 4, which expends the major circuits in ANSNET [1] by inserting additional links to increase the connectivity. For each routing request, the values of s, t, *delay*, *cost*, D and C are randomly generated. The *delay* values of the links are uniformly distributed in the range of $[0..50ms]$, and the *cost* values of the links are uniformly distributed in [0..200]. The performance metric we considered was *success ratio*.

$$success\ ratio = \frac{number\ of\ requests\ successfully\ routed}{total\ number\ of\ routing\ requests}$$

We studied the *success ratio* with respect to $coef$, D and C. The larger the value of $coef$, the higher the probability for the heuristic conditions to be satisfied, which leads to a higher *success ratio*. The smaller the values of D and C, the tighter the constraints of a routing request, which leads to a lower *success ratio*.

The experiment results are presented in Figures 5-9. The x axis represents $coef$ and the y axis represents the *success ratio*. The dimensions of D and C are shown by different figures. Let us take Figure 5 as an example. Each point in the figure is taken by running one thousand randomly-generated routing requests. The values of D and C of all the requests are uniformly distributed in $[100, 115ms]$ and $[400, 460]$, respectively. For the purpose of comparison, we implemented an

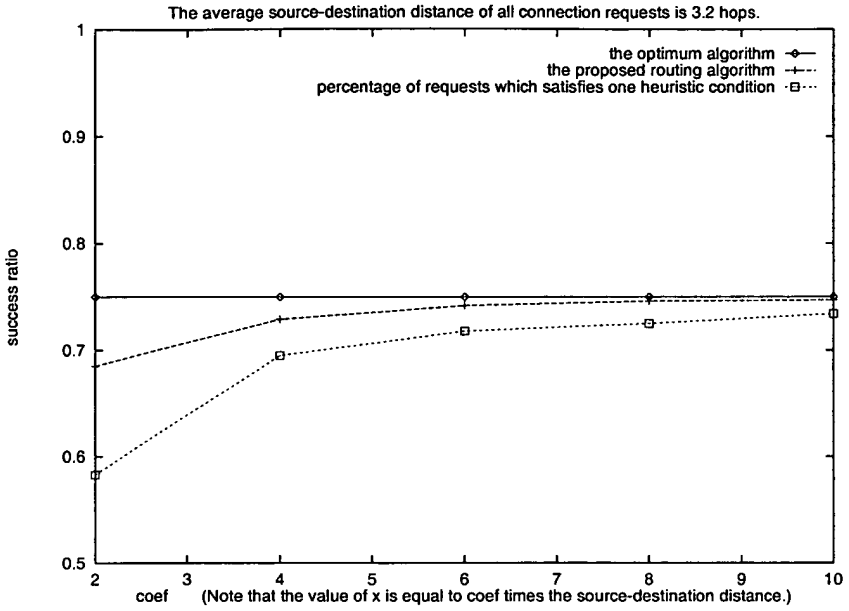


Figure 5: Success ratio with respect to $coef$ when $D \in [100, 115ms]$ and $C \in [400, 460]$. Note that $x = coef \cdot d_{s,t}$. The average distance from the source to the destination of all experimental routing requests is 3.2 hops.

optimum algorithm, which searches all possible paths for a solution with an exponential time complexity. There are three lines in the figure. The highest horizontal line shows the *success ratios* of the optimum algorithm. The lowest line shows the percentage of requests which satisfy one of the *heuristic conditions.* [6] Note that the lowest line is calculated purely based on the heuristic conditions. It gives a lower bound of the performance of the proposed routing algorithm. The actual running performance of the algorithm is given by the middle line, which is constantly above the lowest line. It means that the statistical performance of our algorithm is better than what predicted by the heuristic conditions. Figures 6-9 present the *success ratios* when $D \in [75, 90ms]$ and $C \in [300, 360]$, $D \in [125, 140ms]$ and $C \in [500, 560]$, $D \in [150, 165ms]$ and $C \in [600, 660]$, $D \in [150, 165ms]$ and $C \in [600, 660]$, respectively. Larger values for D and C result in more relaxed delay and cost constraints and thus higher *success ratios* as seen from the figures.

The simulation results show that the *success ratio* of the proposed routing algorithm approaches that of the optimum algorithm when $coef$ is increased. With $coef \geq 4$, the performance of our algorithm is close to that of the optimum algorithm.

[6] The heuristic conditions provide a theoretically-provable performance *estimation* for the algorithm (see Theorem 3). They are useful as a guide line in the design of a network which requires a guaranteed routing performance. However, please be aware that they are *not* part of the routing algorithm.

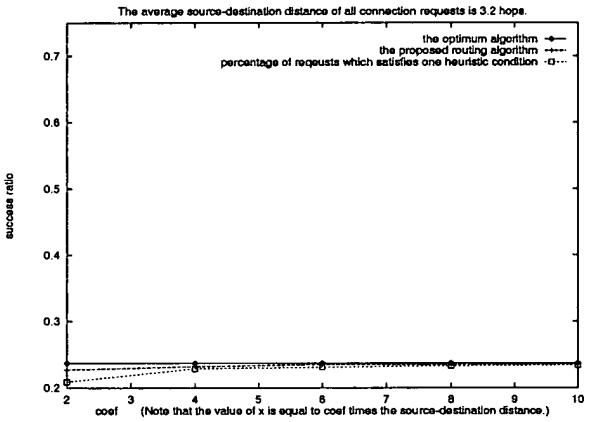


Figure 6: $D \in [50, 65ms]$, $C \in [200, 260]$

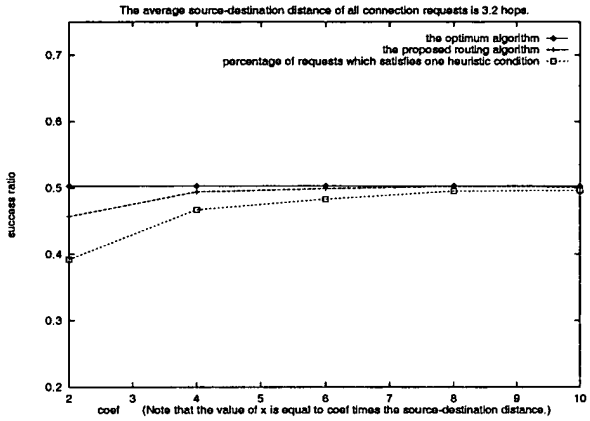


Figure 7: $D \in [75, 90ms]$, $C \in [300, 360]$

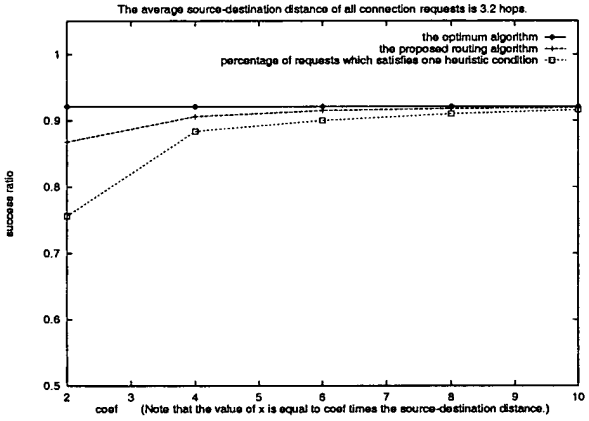


Figure 8: $D \in [125, 140ms]$, $C \in [500, 560]$

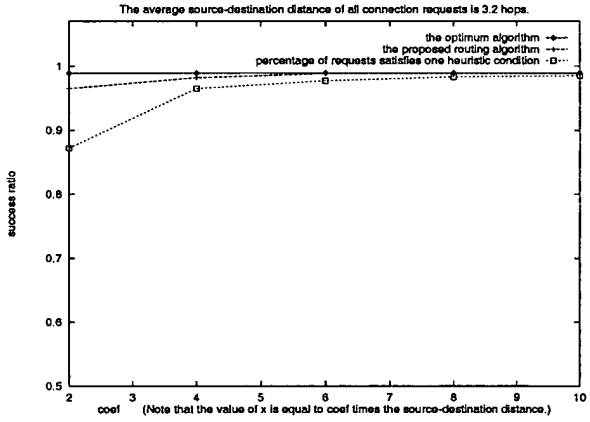


Figure 9: $D \in [150, 165ms]$, $C \in [600, 660]$

3 Related Work

Much work has been done in QoS routing recently [5, 9, 10, 11, 12]. Some routing algorithms consider a single constraint. Plotkin discussed the competitive routing strategy in [9], which considers only the bandwidth requirement. Salama et al [10] proposed a distributed algorithm for the delay-constrained routing. The Sun's algorithm [11] improved the worst-case complexity of the Salama's algorithm. Though both algorithms use heuristic approaches trying to minimize the cost of the found route, the cost is not required to be bounded.

The multi-constrained routing was studied in [5, 12]. Wang and Crowcroft [12] uses the Dijkstra's algorithm in their bandwidth-delay-constrained routing, i.e., finding a path whose bandwidth is not less than a given lower bound and whose delay is not greater than a given upper bound. Note that the bandwidth-delay-constrained routing is not a MCP problem by Definition 1. Both weight functions in MCP, w_1 and w_2, are *additive* — the w_1-*weight* (w_2-*weight*) of a path is equal to the summation of the w_1-*weight* (w_2-*weight*) of all edges on the path. In the bandwidth-delay-constrained routing, however, one of the weight functions, ***bandwidth***, is not additive — the bandwidth of a path is equal to the *minimum* bandwidth of all edges on the path, which makes the problem much easier and can be solved in polynomial time.

Ma and Steenkiste [5] showed that when a class of WFQ-like (Weighted Fair Queuing) scheduling algorithms are used, the problem of finding a path satisfying bandwidth, delay, delay-jitter, and/or buffer space constraints is solvable by a modified version of the Bellman-Ford algorithm in polynomial time. The reason is that when the WFQ-like scheduling algorithms are used, the metrics of delay, delay-jitter and buffer space are no longer independent from each other and all of them become functions of bandwidth, which simplifies the problem and makes it solvable in polynomial time.

All the above algorithms can not solve MCP whose weight functions are assumed to be independent. The work closest to ours was done by Jaffe [4]. Jaffe proposed a distributed algorithm solving MCP with a time complexity of $O(V^5 b \log Vb)$, [7] where b is the largest weight of all edges in the network. The complexity is *pseudo-polynomial* because the run time is polynomial in b, the largest number in the input. See [3] for the definition of the pseudo-polynomial time complexity.

A heuristic algorithm with a polynomial time complexity was also proposed in [4] to approximate

[7] The Jaffe's algorithm finds a solution for *every* pair of nodes in the network whereas our heuristic algorithm does that for a single pair.

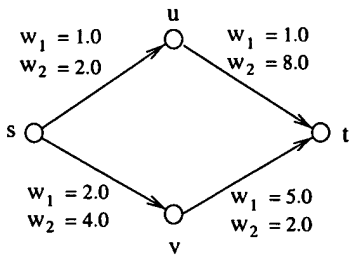


Figure 10: The two constraints are $w_1(p) \leq 8.0$ and $w_2(p) \leq 8.0$. The path, $s \rightarrow u \rightarrow t$, minimizes $w_1(p) + w_2(p)$ but is not a solution. The path, $s \rightarrow v \rightarrow t$, does not minimizes $w_1(p) + w_2(p)$ but is a solution.

the MCP problem. The algorithm, referred to as the Jaffe's approximation algorithm, finds a path which minimizes $w_1(p) + d \cdot w_2(p)$, [8] where d is a given constant. Clearly, it can be implemented by either the Dijkstra's or the Bellman-Ford shortest path algorithm. However, minimizing $w_1(p) + d \cdot w_2(p)$ may not lead to a solution of MCP. Figure 10 gives an example for the case $b = 1$. The problem is MCP$(G, s, t, w_1, w_2, 8.0, 8.0)$. The path, $s \rightarrow u \rightarrow t$, minimizes $w_1(p) + w_2(p)$ but is not a solution because the constraint $w_2(p) \leq 8.0$ is not satisfied. The path, $s \rightarrow v \rightarrow t$, does not minimize $w_1(p) + w_2(p)$ but is a solution. Hence, the Jaffe's approximation algorithm fails in finding a solution in this example. However, our routing algorithm proposed in Section 2.4 finds the solution with $x = 2$.

The time complexity of the Jaffe's approximation algorithm is $O(V^2)$ when the Dijkstra's algorithm is used or $O(VE)$ when the Bellman-Ford algorithm is used. It is much better than the complexity of our heuristic algorithm, which is $O(x^2V^2)$ when EDSP is used or $O(xVE)$ when EBF is used. Therefore, in practice, we can apply the Jaffe's approximation algorithm first. If it fails in finding a solution for MCP, our routing algorithm is executed next.

4 Conclusion

Any multi-constrained routing problem which involves two or more additive weight functions such as delay and cost is NP-complete. We formalized it as an MCP problem (Definition 1) and proposed a heuristic algorithm with a polynomial time complexity. The algorithm first reduces the problem MCP$(G, s, t, w_1, w_2, c_1, c_2)$ to a simpler one MCP$(G, s, t, w_1, w_2, c_1, x)$, and then uses an extended Dijkstra's (or Bellman-Ford) algorithm to find a solution for the new problem in polynomial time.

[8] The paper [4] used different terminologies and notations from ours. $w_1(p) + d \cdot w_2(p)$ was written as $L(p) + d \cdot W(p)$, where L and W correspond to w_1 and w_2, respectively.

We showed the correctness of the algorithm by proving that any solution found for the simpler problem must also be a solution to the original problem. We showed the effectiveness of the algorithm by proving that the simpler problem must have a solution if the original problem has a solution p and $w_2(p) \leq (1 - \frac{l-1}{x})c_2$, where l is the length of p and x is an integer given by the algorithm. With an increasing x, the condition $w_2(p) \leq (1 - \frac{l-1}{x})c_2$ is gradually relaxed and approaching the original constraint, $w_2(p) \leq c_2$. Even when the condition $w_2(p) \leq (1 - \frac{l-1}{x})c_2$ is not satisfied, our algorithm still has a good chance to find a solution because $w_2(p) \leq (1 - \frac{l-1}{x})c_2$ is a sufficient but not a necessary condition. The statistical performance of the heuristic algorithm was studied by experiments, which showed that higher performance of the algorithm can be achieved at the expense of higher overhead.

References

[1] D. E. Comer. *Internetworking with TCP/IP, Volume I, Principles, Protocols, and Architecture.* Prentice Hall, 1995.

[2] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms.* The MIT Press and McGraw-Hill Book Company, 1990.

[3] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* New York: W.H. Freeman and Co., 1979.

[4] J. M. Jaffe. Algorithms for finding paths with multiple constraints. *Networks*, 14:95–116, 1984.

[5] Q. Ma and P. Steenkiste. Quality-of-service routing with performance guarantees. *Proceedings of the 4th International IFIP Workshop on Quality of Service*, May 1997.

[6] Q. Ma, P. Steenkiste, and H. Zhang. Routing high-bandwidth traffic in max-min fair share networks. *Proceedings of SIGCOMM'96*, August 1996.

[7] J. Moy. Ospf version 2, internet rfc 1583. March 1994.

[8] C. Parris, H. Zhang, and D. Ferrari. Dynamic management of guaranteed performance multimedia connections. *Multimedia Systems Journal*, 1:267–283, 1994.

[9] S. Plotkin. Competitive routing of virtual circuits in atm networks. *IEEE Journal on Selected Areas in Communications*, 13:1128–1136, August 1995.

[10] H. F. Salama, D. S. Reeves, and Y. Viniotis. A distributed algorithm for delay-constrained unicast routing. *INFOCOM'97, Japan*, March 1996.

[11] Quan Sun and Horst Langendorfer. A new distributed routing algorithm with end-to-end delay guarantee. *Proceedings of the 4th International IFIP Workshop on Quality of Service*, May 1997.

[12] Z. Wang and Jon Crowcroft. Qos routing for supporting resource reservation. *Journal of Selected Areas in Communications*, September 1996.